METHOD AND APPARATUS OF DEBUGGING COMPUTER PROGRAMS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention is directed to a method and apparatus useful for debugging computer programs and in particular to a method which allows selective tracking of the program by means of command line arguments without the necessity of recompiling the program.

2. Prior Art

When a new computer software product is conceived, there is a typical cycle or process that takes place in the course of bringing the product to the market. The programming cycle typically includes the conception of the idea; design of the software to implement the idea; coding of the program based on the system design; initial testing in the development environment; testing at the user site; and final release of the software to the market.

For example, after an idea occurs for a software product, system design takes place. This includes choosing the language, the compiler and the debugger to use for the product. Thereafter the programmer codes the program based upon the system design. Testing at the development side and in the user environment assures that the program will work as designed. If successful the product is released.

Normally the release of a software product depends on meeting development deadlines. If defects or errors (known as bugs) appear in the code, the product deadline will be missed. This is particularly likely if the bugs are complex, subtle or otherwise difficult to find. Such delays can cause a software product to fail in the marketplace. The present invention provides a debugging tool as a means for meeting deadlines as well as a meand for creating software that goes to market relatively free of errors.

There are many problems with existing debugging software tools. For example, programs are often debugged through the use of print statements which the programmer inserts throughout the program being debugged. When a problem occurs in a program, the programmer inserts the print statements in essentially a hit and miss way in order to try to locate the error. There are several serious problems with this approach.

When the program first fails, there are normally no print statements in the code that would indicate to the programmer where to look for the error. Thus the programmer must either use some separate method to find the general location of the error, or scatter print statements at random throughout the program in the hope that at least one print statement will provide some clues about where the problem lies. Of course, the more subtle the problem, the less likely the programmer is to choose the proper location for a print statement on the first try. Therefore, at the outset, at least, the programmer has no logical place to start the dubugging process.

In order to collect a significant amount of data from which to look for symptoms of the error, the programmer must insert a large number of print statements after the error has occurred. A great deal of time may be spent creating these statements.

Certain kinds of errors change their behavior depending upon the precise location and code. These errors destroy parts of the object code which strongly effects how the errors manifest themselves. For these kinds of errors, inserting print statements may change the nature of the error or even make it seem that the error has disappeared. When the print statements are removed, the error reappears. This kind of error can be extremely frustrating to a programmer trying to track down the ultimate cause of the bug.

The more print statements a programmer uses, the more output is generated. As frequently happens, so much output is generated that any significant information is buried in a mass of unimportant details. Thus, the programmer must always guess whether the benefits of inserting a print statement outweigh the disadvantages of creating unhelpful output.

Inserting print statements requires that all or part of the program be recompiled and relinked which is again a time consuming process. Likewise, when the programmer decides to remove a print statement, the program must be recompiled and relinked again. This also takes time. Once the print statement is removed it may not be reinserted without recompiling. Thus, each insertion or deletion of a print statement requires significant time and effort.

Because of the time required to insert and delete print statements, programmers are reluctant to experiment with output. The programmer is always asking whether the information obtained with a print statement is worth the time involved in inserting and removing the print statement.

It is difficult to keep track of what print statements were used in previous debugging runs. There is no obvious record of print statements that the programmer inserted or removed from one test run to another. This makes it difficult to reproduce and evaluate previous experiments.

When the programmer finds the cause of a bug, the print statements which were inserted must be removed or else they may hide or obscure operation of the program. The programmer must also repeat the entire cycle for each bug encountered.

In summary then, the use of print statements in debugging is extremely time consuming and frustrating for the programmer. It discourages spontaneous experimentation during a development cycle and such experiments if attempted are difficult to reconstruct. It may be impossible to recreate the symptoms of a problem when print statements are inserted and finally all work in finding a bug is discarded once the bug is found. No matter how many bugs a programmer finds, finding the next bug is just as difficult as finding the first. No tools are ever retained, reused or built upon in a logical systematic matter.

SUMMARY OF THE INVENTION

The present invention is a method of debugging a program using machine command line arguments without the necessity of recompiling the program. The method includes the steps of preparing a plurality of macros having corresponding expandable series of source code instructions for selectively tracing the program at selected program locations. The method further includes selectively activating the macros to perform corresponding tracing operations using command line arguments. The macros may be enabled or disabled without affecting the location of program code thereby avoiding the loss of a bug as in prior arrangements. A running count of macro calls and program line executions may be incorporated into the system.